VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2011

Martin Kusyn

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 6. května 2011 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2011 .....

Rád bych na tomto místě poděkoval kolektivu zaměstnanců společnosti RAYNET s.r.o.
za možnost absolvovat praxi v příjemném a inspirujícím prostředí.

Abstrakt

Tato bakalářská práce se zaměřuje na JavaScriptové frameworky a jejich použití, a shrnuje zkušenosti, které jsem získal v průběhu své odborné praxe ve společnosti RAYNET s.r.o. Práce probírá problematiku JavaScriptových frameworků, jejich společné vlastnosti a porovnává nejdůležitější a nejčastěji používané frameworky na trhu. Hlavní část práce popisuje nejdůležitější rysy frameworku ExtJS a jeho rozšíření, které bylo vytvořeno za účelem zjednodušení vývoje CRM aplikací. Tyto technologie jsou použity v aplikaci RAYNET CRM, na jehož vývoji jsem se v průběhu své odborné praxe podílel. Práce také krátce popisuje samotné RAYNET CRM a některé jeho komponenty.

Klíčová slova: ExtJS; JavaScript; Framework; AJAX; CRM

Abstract

This bachelor's thesis is focused on JavaScript frameworks and their usage, and summaries my experience gained during performing my professional practice in the company RAYNET Ltd. Thesis discuss questions of JavaScript frameworks, their commonality and compare the most important and commonly used frameworks on the market. Major part of the thesis describes main features of ExtJS framework and its extension, which was made for simplification of CRM application development. These technologies are used in RAYNET CRM application, in which development I took part during my professional practice. Thesis also briefly describes RAYNET CRM and some of its components.

Keywords: ExtJS; JavaScript; Framework; AJAX; CRM

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript And XML
API	–	Application Interface
BE	–	Back End
BL	–	Business Logic
CRM	–	Customer Relationship Management
CRUD	–	Create, Read, Update, Delete
DOM	–	Data Object Modeling
FE	–	Front End
GUI	–	Graphical User Interface
RIA	–	Rich Internet Application
SAAS	–	Software as a Service
UI	–	User Interface
UX	–	User Experience
XAML	–	Extensible Application Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V průběhu akademického roku 2010/2011 jsem absolvoval individuální odbornou praxi ve společnosti RAYNET s.r.o., jež se primárně zaměřuje na vývoj informačních systémů na míru. Po svém nástupu jsem byl zařazen jako FE vývojář na právě vznikajícím projektu RAYNET CRM, což obnášelo seznámení se s JavaScriptovým frameworkem ExtJS a jeho rozšířením pro zjednodušení vývoje CRM systémů, vyvinutým společností RAYNET.

Jakožto FE vývojář jsem se podílel na vzniku několika UI komponent pro RAYNET CRM. Participoval jsem také na jednom z prvních komerčně nasazených systémů na RAYNET CRM založených. Z důležitějších komponent jsem se podílel mimo jiné na:

Propojení RAYNET CRM se sociální sítí Facebook Toto propojení obnášelo vytvoření adaptéru mezi volně dostupným Facebook Graph API a technologií ExtJS. v rámci adaptéru byly vytvořeny metody zaobalující potřebné metody Facebook Graph API, dále byly vytvořeny aktivní šablony *XTemplate* pro zjednodušené vytváření HTML fragmentů z JSON dat předaných skrz Facebook Graph API, a také několik obslužných UI komponent.

Implementace adaptéru zabrala více než 150 hodin práce, během kterých byl adaptér jak implementován, tak odladěn pro potřeby RAYNET CRM.

Vytvoření uživatelské nástěnky – dashboardu Nástěnka v aplikaci RAYNET CRM nabízí uživateli přehled preferovaných prvků. Funkčnost a vzhled nástěnky si uživatel sestavuje z nabízených komponent sám. Komponenta nástěnky je implementována zahrnutím UX rozšíření *PortalColumnLayout*, *Portlet*, *PortletProvider* a *HttpStateProvider*. *PortalColumnLayout* je rozšíření umožňující pozicování panelů v rámci sloupců, *Portlet* je rozšířením panelů pro lepší spolupráci s *PortalColumnLayout*, *PortletProvider* je interně vyvinutá komponenta zajišťující lazy loading JavaScriptových souborů se zdrojovými kódy jednotlivých *Portletů*. Implementací *HttpStateProvideru* je zajištěno ukládání rozmístění a nastavení jednotlivých Portletů na server místo do cookies soborů na pevném disku počítače.

Implementace *dashbardu* a její odladění bylo otázkou 150-200 hodin práce na FE. Práce navíc vyžadovala vytvoření podpůrných tříd na BE straně aplikace.

Funnel Chart *Funnel chart* je graf ve tvaru obrácené pyramidy mající za úkol zobrazit úbytky měřené entity v rámci jednotlivých stádií. Při vývoji bylo potřeba stanovit matematický model pro vykreslení grafu – z několika dříve uvažovaných variant byla nakonec vybrána varianta založená na plošné reprezentaci grafu.

Funnel chart byl vtvořen již na nové verzi ExtJS frameworku, konkrétně na verzi ExtJS 4.0, a byl upraven tak, aby byl kompatibilní s grafy obsaženými přímo v rámci samotného ExtJS 4.0. Aby mohl být graf provozován pod aplikací RAYNET CRM, je třeba propracovat ještě jeho funkčnost v *sandbox módu*.

Vytvoření matematického modelu a samotná implementace *Funnel chartu* zabrala kolem 80 hodin práce. Další čas byl věnován začištění komponenty a prototypu *sandbox módu*.

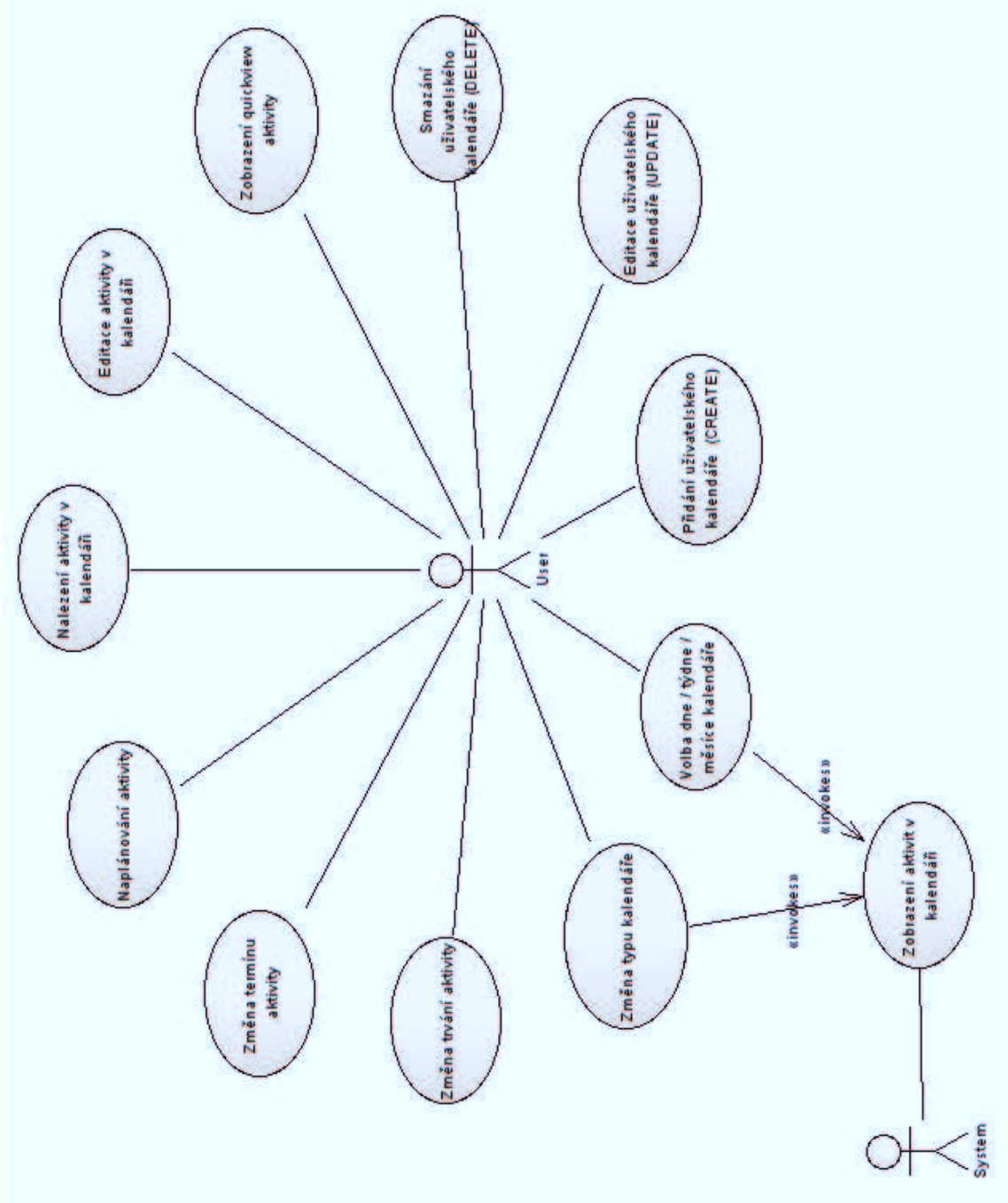
Implementace plánovacího kalendáře V rámci praxe jsem byl také pověřen implementací plánovacího kalendáře na projektu pro externího zákazníka. Pro implementaci samotnou bylo potřeba nejprve osvojení si rozšíření Bryntum Ext-Scheduler pro ExtJS a následně jeho přizpůsobení pro potřeby správného zobrazení dat několika druhů entit.

Vývoj komponenty plánovacího kalendáře zabral více než 300 hodin práce, během kterých byl prvně vyvinut prototyp a následně byl přizpůsoben pro komunikaci s BE stranou aplikace. Komponenta je již nasazena, ale stále je upravována a rozšiřována podle představ zákazníka.

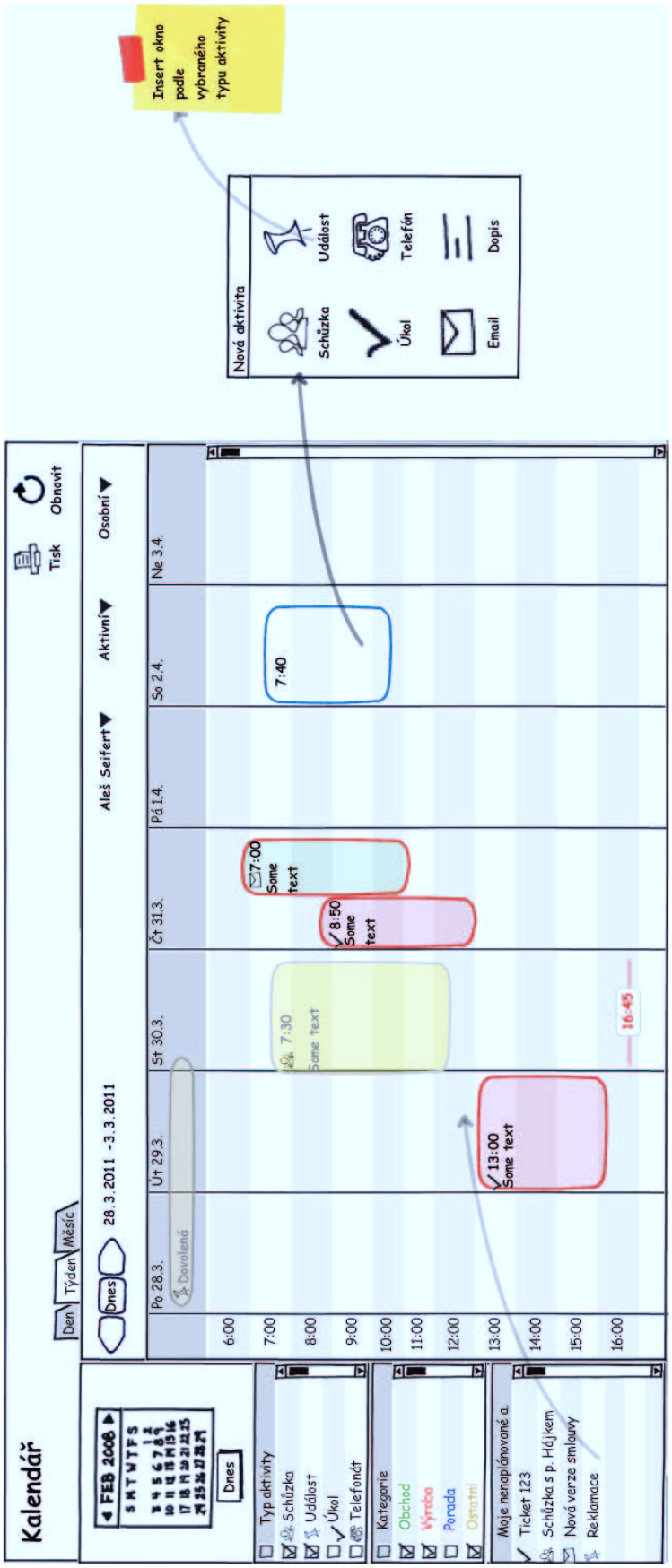
Kalendářová komponenta RAYNET CRM Kalendář je důležitou komponentou RAYNET CRM, která má za účel zobrazování a plánování aktivit jak jednotlivých uživatelů, tak celých skupin. Implementace kalendářové komponenty je realizovaná integrací rozšíření Ext.ensible Ext Calendar Pro pro ExtJS. Kalendářová komponenta je stále ve fázi prototypu, jehož účelem je ověřit možnosti zmíněného Ext Calendar Pro. Během vytváření prototypu jsem komunikoval s výrobcem rozšíření a podílel jsem se také na jeho lokalizaci.

Vývojem kalendářové komponenty se zaobírám právě ve dnech sepisování této práce a právě proto bych na ní rád demonstroval ukázky vstupů a zadání výroby.

Již existující komponentu Ext Calendar Pro je třeba upravit tak, aby byla schopna korektně pracovat se zjednodušenými objekty aktivit, zároveň je třeba upravit proces vytváření záznamů v kalendáři tak, aby se po vytvoření záznamu tahem zobrazila uživateli nabídka s výběrem typu aktivity. Po výběru typu aktivity bude uživateli zobrazeno okno pro vložení konkrétní aktivity. Dále je potřeba přizpůsobit komponentu tak, aby umožňovala přetažení již předdefinované aktivity do kalendáře z externího panelu, a bylo tak určeno její časové zařazení.



Obrázek 1: Use case diagram použití kalendářové komponenty



Obrázek 2: Návrh UI kalendářové komponenty

2 Problematika JavaScriptových frameworků

2.1 Frameworky

V dnešní době jsou při programování hojně využívány frameworky. Jedná se komplexní části kódu nabízející funkcionalitu zaštiťující řešení známých problémů. Rozsah frameworku se může lišit, některé frameworky nabízejí pouze zjednodušený přístup k určenému API, jiné mohou nabízet komplexní soubory knihoven, implementaci často používaných návrhových vzorů či UX prvky.

Existence frameworku umožňuje programátorovi zaměřit se na řešení konkrétního problému, místo implementace již známých a hojně používaných postupů. Vhodnost použití frameworku je vždy dána jeho nabízenou funkcionalitou a potřebami daného projektu. Nasazení frameworku znamená složitější strukturu projektu a zvýšení nároků na výkon HW, vyšší míra abstrakce naopak přináší výhody znovupoužitelnosti kódu. Častou námitkou proti použití frameworku je delší doba osvojení frameworku.

Frameworky jsou v dnešní době používány ve většině dnes používaných programovacích jazyků, od jazyků primárně určených pro vývoj desktopových aplikací až po aplikace serverové. Výborným příkladem vhodného použití jsou dynamické webové aplikace, často využívající JavaScriptových frameworků.

Použití samotného JavaScriptu bylo velmi problematické z důvodů mírných odlišností jednotlivých webových prohlížečů, používajících různá JavaScriptová jádra. Většinou to mělo za následek špatnou kompatibilitu kódu a aplikace musela být buďto zjednodušena, nebo složitě a dlouho laděna. Účelem většiny dnešních JavaScriptových frameworků je právě sjednocení požadované funkcionality napříč prohlížeči, tzn. ošetření, aby se kód napsaný v daném frameworku choval korektně jak v Internet Exploreru, tak ve Firefoxu, Chrome nebo Safari.

2.2 Důvody pro použití JavaScriptových frameworků

S rozšířením vysokorychlostního neomezeného připojení k internetu došlo k několika zásadním změnám v pohledu na web. Webové stránky se staly mnohem rozsáhlejšími, dynamičtějšími a nabídly podstatně širší funkcionalitu. Se zlepšením stability připojení se začaly na internet migrovat také služby do té doby desktopové. Platformy využívající tyto služby nazýváme RIA (Rich Internet Application). Účelem RIA aplikací je poskytnout uživateli komfort desktopových aplikací v rámci webového prohlížeče nezávisle na platformě[6].

Důležitým rysem RIA aplikací je dynamika změny obsahu bez potřeby kompletního překreslení webové stránky. V době vzniku prvních RIA aplikací bylo potřeba se oprostit od omezení HTTP protokolu a jeho modelu žádost/odpověď a s tím souvisejícím překreslením. Požadovaná data bylo potřeba získávat na pozadí aplikace a aktivně měnit pouze část stránky určenou pro jejich vykreslení[8].

Začaly se hledat cesty jak tyto požadavky splnit a objevily se technologie, které potřebné změny umožňovaly – například Adobe Flex nebo XAML. Tyto technologie byly však do značné míry závislé na platformě, na které byly provozovány. Jako jedna z nej-

schůdnějších cest se tehdy ukázalo použití JavaScriptu, který do té doby nebyl příliš vhodný pro seriózní nasazení. Pomalu ale jistě se začalo mluvit o AJAXu.

AJAX samotný není konkrétní technologií (přestože existují obecné implementace), jedná se spíš o obecný koncept zastřešující požadavky RIA aplikací. Mezi technologie se kterými se setkáváme při implementaci AJAXu řadíme JavaScript, DOM a XHR. Logika používání AJAXu je zhruba následující – jako odpověď na činnost uživatele v GUI je spuštěna JavaScriptová metoda z AJAX Enginu (implementace AJAX konceptu), který vytvoří a vyvolá *XHR request*, jímž dotáže požadovaná data ze serveru, a následně je pomocí metod pro manipulaci s DOM zapíše tam, kde by se měla zobrazit[4].

Vývojáři komplexnějších aplikací na JavaScriptu se většinou potýkají s podobnými problémy. Místo toho, aby se věnovali kódu samotné aplikace musí mnohdy řešit rozdíly mezi prohlížeči, které ovlivňují chod jejich aplikace tak i její grafickou prezentaci. Také samotná implementace AJAXu se neřadí mezi primitivní záležitosti. Z těchto důvodů začaly vznikat JavaScriptové frameworky. Jejich snahou je zaštítit funkcionalitu AJAXu a skrze své API oprostit vývojáře od řešení rozdílnosti jednotlivých platforem.

2.3 Standardní funkcionalita JavaScriptových frameworků

Základní požadavky při vývoji moderní webové aplikace se většinou neliší, proto frameworky obvykle nabízejí velmi podobnou funkcionalitu. Obvykle tedy frameworky implementují následující:

- Selektory
- Průchod DOM stromem
- Manipulaci s DOM
- Rozšíření funkcionality
- Zpracování událostí
- AJAX
- Prvky UX

Tyto možnosti nabízí většina frameworků ať už se zaměřují spíš na vykreslování grafiky, nebo komunikaci na pozadí.

Selektory Při vytváření dynamické aplikace mnohdy potřebujeme vybrat element DOM struktury tak, abychom s ním mohli pracovat (měnit pozici, styl, obsah). Už samotný JavaScript umožňuje vybrat element pomocí jeho id

```
var el = document.getElementById('elementId');
```

Frameworky jdou obvykle dál a umožňují vybírat element nejen podle jeho id, ale také podle jeho vlastností či aplikovaných CSS stylů.

Průchod DOM stromem Ne vždy lze vybrat element DOM objektu pomocí jeho id nebo aplikovaných stylů. může nastat případ že potřebujeme vybrat rodiče DOM elementu, který právě máme k dispozici, nebo potomky. Pro takové případy frameworky implementují metody pro průchod DOM stromem podle zadaných kritérií.

Manipulace s DOM S vybraným DOM elementem můžeme dále pracovat – přepisovat jeho obsah, přidávat data k jeho obsahu, přidávat nebo odebírat styly, které jsou na tento element aplikovány.

Rozšíření funkcionality JavaScript samotný mnohdy nenabízí funkce, na které je programátor zvyklý z vyšších programovacích jazyků určených pro desktopové programování. Příkladem takového rozšíření je třeba funkce *foreach*, která má za účel projít všechny prvky pole.

Zpracování událostí Zpracování událostí je klíčové z hlediska přiblížení chování aplikace k chování známému z desktopových aplikací. Tím že framework zvládne monitorovat a obsluhovat události uživatelského rozhraní jsou položeny základy pro další rozšíření funkcionality, například o *Drag&Drop*.

AJAX Podstatnou součástí frameworku je implementace AJAXu, který umožňuje asynchronní komunikaci se serverem a získávání pouze požadované části dat.

UX/UI rozšíření Většina frameworků také nabízí implementaci komponent uživatelského rozhraní at' už formou přímé implementace nebo formou pluginu . Právě přítomnost UI komponent je tím, čím se od sebe většina frameworků liší.

2.4 Porovnání nejrozšířenějších Javascriptových frameworků

2.4.1 Dojo toolkit

http://dojotoolkit.org

Dojo toolkit je open source modulární JavaScriptová knihovna, která vznikla za účelem usnadnění vývoje JavaScriptových aplikací. Základem Dojo toolkitu je knihovna nejčastěji používaných API a funkcí, ale Dojo také nabízí různá rozšíření uživatelského rozhraní obsahující nejčastěji používané UI prvky.

Dojo toolkit je distribuován pod dvojí licencí – BSD a ALF. v současnosti je dostupná verze 1.6.0.

2.4.2 jQuery

http://jquery.com/

jQuery je malá knihovna funkcí zpřístupňující manipulaci s DOM, AJAX a ošetření událostí. Samotné jQuery nabízí vcelku základní funkčnost a počítá s využitím rozšíření,

která jsou pro něj dostupná. To se týká například UI prvků, které jQuery samotné neimplementuje, ale mnoho základních UI prvků je dostupných v rozšíření jQuery UI.

jQuery je distribuována pod dvojí licencí GPL a MIT. v současnosti je dostupná verze 1.5.

2.4.3 YUI

http://developer.yahoo.com/yui/

YUI, neboli Yahoo! User Interface Library, je JavaScriptová knihovna vyvíjená týmem Yahoo! určená přímo pro vývoj RIA aplikací, čemuž odpovídá i bohatá nabídka UI prvků implementovaná přímo v samotné knihovně.

YUI je distribuovaná pod licencí BSD a v současnosti je dostupná ve verzi 3.3.0.

2.4.4 MooTools

http://mootools.net/

MooTools, neboli My Object-Oriented Tools je modulární knihovna rozšiřující funkčnost JavaScriptu. Knihovna je rozdělena a pro aplikace se načítá pouze ta část knihovny, která je pro vytváření aplikace potřeba. MooTools se primárně zaměřují na rozšíření JavaScriptu o pohodlnou práci s objekty, ale nabízí také rozšíření pro samotnou práci s uživatelským rozhraním. v tomto směru je však knihovna relativně chudá a v případě potřeby je třeba dohledávat pluginy třetích stran.

Čím se MooTools výrazně liší od ostatních JavaScriptových frameworků je zaměření na rozšíření samotného JavaScriptu. Zatímco se většina ostatních frameworků snaží zapouzdřovat objekty tak, aby nehrozilo nebezpečí kolizí se skripty třetích stran. MooTools naopak patří mezi frameworky přímo rozšiřující nativní objekty tak, aby poskytl uživateli frameworku co nejvyšší komfort.

MooTools je distribuovaná pod licencí MIT a v současnosti je dostupná ve verzi 1.3.

2.4.5 Prototype JS

http://prototypejs.org/

Prototype vznikl původně jako základ AJAX podpory pro Ruby on Rails. Podobně jako MooTools se zaměřuje na rozšíření objektově orientovaného modelu samotného JavaScriptu. Prototype JS jako takový nabízí vcelku základní funkcionalitu, ale je mnohdy zahrnut jakou součást větších projektů jako je již zmíněné Ruby on Rails, nebo Rico či script.aculo.us.

Prototype JS je distribuovan pod licencí MIT a v současnosti je dostupná ve verzi 1.7.

3 ExtJS framework

ExtJS je javascriptový framework stvořený přímo pro vývoj RIA aplikací. Čím se od ostatních dostupných frameworků liší je jeho rozsáhlost. ExtJS nabízí krom prvků známých z většiny ostatních frameworků také podporu lokálních úložišť, grafické efekty a velké množství UI prvků. ExtJS navíc poskytuje podporu všech primárních prohlížečů[7].

3.1 Ext Core

Ext Core je základem ExtJS frameworku. Oproti plné verzi balíku nabízí Ext Core základní metody pro tvorbu dynamických webových stránek. Vzhledem k tomu že Ext Core neobsahuje žádné prvky uživatelského rozhraní, je vhodná převážně pro méně náročné aplikace[3].

3.1.1 Funkcionalita Ext Core

Rozšíření JavaScriptu Ext Core rozšiřuje JavaScript o lepší funkcionalitu pro práci s poli, řetězci a funkcemi. Pro práci s poli nabízí Ext Core metody `indexOf`, zjišťující zda je objekt v poli obsažen, a `remove` pro odstranění objektu z pole. Práci s řetězci rozšiřuje Ext Core o metodu `format` umožňující formátování řetězce.

Důležité je rozšíření práce s funkcemi, které Ext Core nabízí. Každé funkci je nyní pomocí funkce `createCallback` možné vytvořit volání této funkce, které je vhodné například pro vykonání funkce po kliknutí na tlačítko. Jako příklad můžeme uvést vyvolání okna s upozorněním[1].

Dále jsou k dispozici funkce `defer`, pro odložení vykonání funkce o zadaný čas, `createDelegate`, umožňující nastavit *scope* konkrétního objektu, a `createInterceptor`, pomocí které lze vytvořit funkci předchůdce[2].

Globální funkce Ext Core Samotné Ext Core dále rozšiřuje práci s objekty. Pomocí funkcí `Ext.isArray`, `Ext.isEmpty`, `Ext.isFunction`, `Ext.isObject` a `Ext.isPrimitive` můžeme zjistit jakého je objekt typu. `Ext.apply` a `Ext.applyIf` slouží pro rozšiřování objektů, kde funkce `Ext.apply` nahradí hodnoty původního objektu a funkce `Ext.applyIf` do objektu přidá pouze hodnoty, které v něm dosud nejsou zavedeny.

Funkce `Ext.encode` a `Ext.decode` slouží k převedení JavaScriptového objektu na JSON a zpět. Tyto funkce jsou pouze zkratkami pro volání funkcí encode a decode z balíku `Ext.util.JSON`, sloužícího pro práci s JSON objekty.

Mezi globální funkce se také řadí základní funkce pro práci s DOM, tedy funkce `Ext.get`,`Ext.getBody`,`Ext.getDom`,`Ext.query`,`Ext.removeNode` a `Ext.select`. Zajímavou je v tomto směru funkce `Ext.fly` která podobně jako `Ext.get` slouží k výběru elementu na základě elementu HTML, oproti `Ext.get` však nevrací samotný element ale pouze pointer na úložiště, kde je tento element dočasně uložen.

[1] Příklad použití viz. Výpis 1 v příloze B

Důležité je zavedení tříd a jmenných prostorů – namespaců. `Ext Core` nabízí funkce `Ext.namespace` pro vytvoření nového jmenného prostoru, `Ext.extend` zajišťující dědičnost a `Ext.override` pro rozšíření původní třídy o novou funkcionalitu.

Ext.util.Observable Třída `Ext.util.Observable` rozšiřuje `Object` a zajišťuje rozhraní pro publikování událostí. Třída obsahuje metody pro registraci handler funkcí k jednotlivým událostem, stejně jako jejich odpalování a odregistrování.

`Ext.util.Observable` je třída natolik zásadní, že je děděna většinou tříd balíku ExtJS.

Ext.data.Connection `Ext.data.Connection` je třída zapouzdřující připojení k doméně, ze které je stránka nahrána. URL může být definována jak při vytvoření připojení, tak před samotným odesláním požadavku. Zasílání požadavků probíhá asynchronně, pro ošetření odpovědi je tedy třeba požít callback funkce.

Ext.Ajax `Ext.Ajax` je singleton třída rozšiřující `Ext.data.Connection` a poskytuje rozraní pro pohodlnou práci s XHR[2].

Ext.util.TaskRunner a Ext.TaskMgr Třída `Ext.util.Taskrunner` se stará o periodické spouštění funkcí po daném časovém intervalu. Pomocí této třídy tak lze např. nahradit timer(časovač) – v kombinaci s odpalováním událostí. `Ext.TaskMgr` je statická instance třídy `Ext.util.TaskRunner` používaná pro spouštění doplňkových služeb.

Ext.Fx `Ext.Fx` poskytuje základní podporu visuálních efektů. Krom několika efektů přináší také možnost tyto efekty řetězit a vytvořit tak základní animace elementů.

Ext.Element Třída `Ext.Element` zapouzdřuje DOM element a normalizuje rozdíly mezi jednotlivými prohlížeči. Všechny instance této třídy dědí metody třídy `Ext.Fx`, umožňující vizuální efekty nad všemi DOM objekty.

Události této třídy nejsou běžnými událostmi ExtJS, ale zapouzdřují události prohlížeče (např. *blur*, *focus*, *click*, *dblclick*, *keydown* aj.)

Třída poskytuje metody pro práci se samotným HTML elementem – pozicování, nastavení CSS stylů, přidávání a odebírání CSS tříd, získání rodiče, přidávání a odebírání potomků a mnoho dalších.

Ext.EventObject Podobně jako `Ext.Element` zapouzdřuje nativní DOM elementy, zapouzdřuje `Ext.EventObject` události prohlížeče a normalizuje je napříč platformami. Obsahuje také mechanismy na potlačení propagace událostí tak, aby nebyly vyvolány defaultní akce. Handler funkce k těmto událostem jsou registrovány v `Ext.EventManager`.

[2]Příklad použití viz. Výpis 2 v příloze B

Ext.Template `Ext.Template` umožňuje vytvoření šablony pro generování HTML fragmentů na základě dosazených hodnot.

3.2 ExtJS

Kompletní balík ExtJS zahrnuje veškerou funkcionalitu Ext Core a dále ji rozšiřuje například o datové modely a lokální úložiště, komponenty UI, layouty a jiné utility použitelné při vývoji RIA aplikací[1].

3.2.1 Datový model a lokální úložiště

ExtJS přináší podporu lokálních úložišť – *store*. Pro *store* je typická jasná datová struktura, založená na předem definovaných typech záznamů – *record*.

Instance třídy `Ext.data.Record` zapouzdřují jak definici záznamu, tak i jeho hodnoty. Konstruktory těchto tříd jsou generovány předáním pole definicí atributů (`Ext.data.Field`) a instance jsou obvykle vytvářeny instancí třídy `Ext.data.Reader`, která zpracovává nenaformátované datové objekty.

U *recordu* je důležité, že jeden záznam patří pouze do jednoho *store*, lze mezi *story* přesunovat ale v případě potřeby je třeba zavolat metodu pro vytvoření kopie *recordu*. *Record* samotný také referencuje *store* kterému náleží. To je potřeba si uvědomit hlavně před uložením záznamu na server – *record* samotný by neměl být serializován, ukládat by se měla jeho položka data, obsahující záznamy odpovídající definici *recordu*.

Samotný *store* je instancí třídy `Ext.data.Store` nebo některé ze tříd z ní odvozených. ExtJS poskytuje několik konkrétních store tříd – například `ArrayStore` pracující nad polem, `DirectStore` poskytující základní definici pro store, `GroupingStore` který umožňuje seskupování záznamů podle kategorií, `JsonStore` a `XmlStore`, které zjednodušují práci s daty ve formátu JSON/XML. Pokud by žádná z nabízených konfigurací neposkytovala potřebnou funkcionalitu, lze samozřejmě vytvořit store vlastní.

Při sestavování *store* je třeba definovat tři jeho základní komponenty – *proxy*, *reader* a *writer*. *Proxy* se stará o komunikaci se serverem, nese informace o všech potřebných URL pro CRUD operace a parametrech které budou při požadavku odesílány. *Reader* má na starosti mapování příchozích dat do recordu, které jsou do storu vkládány a *writer* naopak obstarává přepis dat do formy posílané serveru.

Jednotlivé předdefinované konfigurace již mají definovány jaké komponenty budou použity. Kupříkladu `JsonStore` má předdefinován `JsonReader`, `ArrayStore` defaultně používá `ArrayReader`. Pokud komponenty nevyhovují, lze je ve většině případů nahradit.

Store navíc poskytují metody pro třídění, filtrování či vyhledávání. Dále je také možné provést *rollback* změn, pokud data ještě nebyla odeslána na server – v praxi je většinou store nastavený tak, aby s uložením provedl *commit* a odeslání dat na server.

Store se při vytvoření registrují do `Ext.StoreMrg`, třídy držící reference na jednotlivé story – ty lze vyhledávat buď podle id, nebo podle indexu přiřazeném ve `StoreMgr`.

3.2.2 Prvky UI

Plnohodnotné ExtJS obsahuje velké množství prvků uživatelského rozhraní. Cílem ExtJS bylo přinést do světa webových aplikací prvky známé z desktopových aplikací – ExtJS tedy obsahuje většinu uživatelských prvků se kterými se setkáváme v jazycích určených pro desktopové programování. Přítomny jsou jak jednoduché formulářové prvky jako jsou *checkboxy*, *comboboxy*, textová a číselná pole, *radiobuttony* či datové pole, tak předpřipravené komponenty – například tabulky, *datepickery*, textové editory, kontextová menu, výběrové stromy, palety barev, okna atd.

ExtJS také definuje *layouty*, určující rozložení prvků na stránce, ale i v jednotlivých panelech. Přítomny jsou *layouty* pro formuláře, ale také *záložkové layouty*, *wizardy*, vertikální a horizontální boxy, sloupce a další. ExtJS lze samozřejmě rozšířit o další *layouty* – zajímavým rozšířením je například `PortalColumnLayout`, který v několika sloupcích nese panely s obsahem – *portlety*. *Portlety* lze mezi sloupci libovolně přemisťovat, zmenšovat, odebírat či přidávat. Tohle je vhodné například pro vytvoření nástěnky uživatele v komplexnějších aplikacích – viz. nástěnka RAYNET CRM.

Základem uživatelského rozhraní v Ext JS je třída `Ext.Component` definující základní chování komponent – funkce které by komponenty měly podporovat, události které by měly odpalovány. Ve třídě je zabudována podpora pro základní chování aktivní/neaktivní a zobrazený/schovaný. Všechny komponenty jsou při vytváření registrovány do `Ext.ComponentMgr`, čímž je umožněno vybírat komponenty pomocí metody `Ext.getCmp`, které je předáváno id této komponenty.

Důležitá je pro komponenty možnost zaregistrovat svůj *xtype* – pokud je pak komponenta přidávána například jako položka panelu, není nutno tuto komponentu předem vytvářet, ale uvést panelu její konfiguraci obsahující její *xtype* a panel se při renderování postará o samotné vytvoření této komponenty.

Třída `Ext.Component` je dále rozšířená třídou `Ext.BoxComponent`, která tvoří základ všem komponentám pracujícím jako box s danou výškou a šířkou. `Ext.BoxComponent` nabízí metody korektní pozicování v rámci renderovacího modelu.

`Ext.Container` je rozšíření třídy Ext.BoxComponent o možnost obsahovat další komponenty. Veškeré komponenty, u kterých se předpokládá že by mohly obsahovat jiné komponenty (panely, menu, okna) tak tedy vychází ze třídy `Ext.Container`, zatímco jednoduché prvky jako tlačítka, textová pole či *comboboxy* dědí ze třídy `Ext.BoxComponent`. Třída obsahuje metody pro práci s vnitřními komponentami – jejich přidávání, vkládání a odebírání. Reference na veškeré vlastněné komponenty jsou uloženy v poli items.

Při vytváření kontejneru lze nastavit defaultní vlastností jeho vlastněných komponent. Komponentám lze nastavit defaultní *xtype*[3]), ale také jiné vlastnosti. Pokud nebude komponenta definována jinak, budou na ní tyto vlastnosti aplikovány. Přidávání komponent do kontejneru lze provádět jak přes referenci na již existující objekt, tak pomocí konfiguračního objektu. V druhém případě se na komponenty aplikují nastavené defaultní vlastnosti.

[3]Příklad použití viz. Výpis 3 v příloze B

3.3 ExtJS 4.0

Koncem dubna 2011 byla představena již čtvrtá verze ExtJS frameworku, která mimo jiné zjednodušuje datový model, zlepšuje výkonnostní optimalizace a přichází s nativní podporou vektorových grafických enginů SVG a VML, pomocí nichž jsou v ExtJS 4.0 také nově implementovány vysoce přizpůsobitelné interaktivní grafy. Odpadá tak nutnost kombinovat JavaScript s technologií Flash, čímž se také zlepšuje použitelnost na mobilních zařízeních.

ExtJS 4.0 podporuje rovněž *sandboxový mód*, s jehož pomocí lze provozovat na jedné stránce souběžně ExtJS 3.3 a ExtJS 4.0.

4 CRM rozšíření ExtJS frameworku

Firma RAYNET s.r.o. se momentálně mimo jiné zabývá vytvářením RAYNET CRM.

CRM neboli Customer Relationship Management (Řízení vztahu se zákazníky) jsou databázové technologie zaměřené na shromažďování, zpracování a využití dat o zákaznících firmy. CRM má za úkol zpřehlednit proces komunikace se zákazníkem[5].

4.1 Rozšíření pro vývoj CRM

Pro usnadnění FE části vývoje RAYNET CRM a pro dosažení potřebné funkcionality, bylo potřeba rozšířit zvolenou technologii (Ext JS), tak aby plně vyhovovala požadavkům pro vývoj aplikace takového rozsahu. Krom rozšíření UI komponent, bylo potřeba také vyřešit zabezpečení aplikace, lazy loading knihoven, ošetření událostí. Za tímto účelem byl v rámci aplikace registrován namespace Crm, ve kterém jsou implementovány potřebné změny a rozšíření.

4.1.1 Rozšíření funkcionality frameworku

Crm.app.EventBroker `EventBroker` má na starost ošetření událostí na globální úrovni – komponenta tedy nemusí referencovat komponentu na jejíž události potřebuje reagovat, stačí když se přihlásí k odběru událostí do `EventBrokeru`. Prvním krokem je registrace topicu (tématu) pomocí funkce `Crm.EB.registerTopic`. Poté co je topic registrován lze k němu pomocí funkce `Crm.EB.subscribe` registrovat metody ošetřující jeho odpálení pomocí metody `Crm.EB.broadcast`. Dále lze metody od tématu odregistrovat pomocí `Crm.EB.unsubscribe` a odregistrovat samotný topic pomocí `Crm.EB.unregisterTopic`.

Crm.app.ScriptLoader `ScriptLoader` se stará o lazy loading skriptů. Ne všechny skripty jsou potřeba při zavádění stránky a jejich nahráváním se jen prodlužuje čas potřebný k vykreslení pracovního prostředí. je tedy vhodné odložit nahrání těchto skriptů až do momentu kdy budou potřeba. Tuto funkcionalitu zajišťuje právě `ScriptLoader`. Aktivace `ScriptLoader` probíhá zavoláním funkce `Crm.LoadScript` jejímž parametrem jsou objekty nesoucí v poli informace o souborech, které jsou požadovány k nahrání, callback funkci, která bude vyvolána po jejich nahrání a scope ve kterém tato funkce proběhne.

Při spuštění `ScriptLoaderu` dojde ke kontrole, zda již nejsou dané soubory již nahrány – pokud ano, ukončí se nahrávání a je rovnou vyvolána callback funkce. Pokud ale soubory nahrány nejsou, provede se pro každý soubor požadavek na získání obsahu souboru se skriptem, který je následně vložen do HTML souboru aplikace a zaregistrován `ScriptLoaderem` pro případ, že by byl tento soubor znovu dotazován. Poté je provedena callback funkce.

Crm.app.ErrorManagement `Crm.app.ErrorManagement` je rozšířením `Ext.data.Connection` o globální ošetřování výjimek. Při přijetí odpovědi od

serveru se provede kontrola této odpovědi – server může vrátit odpověď se stavem 200, ale přitom vracet upozornění místo korektních dat. v případě, že nevrací korektní odpověď provede se prověření zda je uživatel přihlášen a chyba se propaguje dále.

Crm.adapter.FB `Crm.adapter.FB` zajišťuje konektivitu se sociální sítí Facebook. Adaptér zapouzdřuje funkcionalitu JavaScriptového API poskytovaného Facebookem. Umožňuje přihlášení se k síti, stahování informací o uživatelích (samozřejmě pouze v rámci práv přihlášeného uživatele), sledování zdí uživatelů, stránek a skupin, komentování a publikování atd. Adaptér také obsahuje několik `XTemplate` šablon pro vytvoření HTML fragmentů pro výpis přijatých dat.

Crm.app.Helper `Crm.app.Helper` poskytuje pomocné metody pro práci s aplikační hierarchií a bussines logikou. Obsaženy jsou metody pro otevírání entitních pohledu bussines logiky (pro každou entitu existuje sada pohledů pro vkládání, zobrazení detailu, zobrazení v seznamu, zobrazení v katalogu, náhled pro vyhledávání atd.). Zobrazení samotného pohledu pak probíhá jednoduše zavoláním příslušné funkce s argumenty určujícími entitu a její id – např. zavolání `Crm.app.Helper.openDetail('Osoba',1)` zavolá vytvoření a otevření panelu z konkrétní implementace DetailView pro entitu Osoba a přiřadí jí id 1, podle kterého budou ze serveru dotaženy data vztahující se k záznamu 1 entity Osoba.

4.1.2 Entitní pohledy

Pro práci s objekty business logiky je třeba pro každou entitu vypracovat pohledy na tuto entitu. Přestože se konkrétní pohledy pro každou entitu liší, vycházejí ze společných základů.

Crm.view.InsertView `InsertView` je pohled pro vytváření nových záznamů. Třídy odvozené z obecného `InsertView` definují `recordSpecification`, který nese informace o entitě a jejích položkách, se kterými se bude při vkládání pracovat – `InsertView` nemusí pracovat se všemi daty záznamu z databáze, pro vytvoření stačí pouze povinné záznamy. Na základě `recordSpecification` se vytváří `Crm.data.InsertViewStore`, který nese data formuláře. Dále odvozené třídy obsahují specifikaci uživatelských prvků formuláře – krom běžných prvků jako jsou textová pole může formulář také obsahovat `ReferenceField`, kterým se vybírají reference na záznamy jiné entity. `Crm.view.InsertView` pak poskytuje metody pro *bindování* dat formuláře. Vyplněná data z formuláře jsou pomocí `dataIndexu` *bindována* do *storu* a jejich ukládání probíhá klasicky uložením *storu*.

Crm.view.ListView `ListView` je základní pohled pro přehled záznamů entity, obsahuje jen ty nejzákladnější informace. Podobně jako třídy odvozené od `InsertView` definují i třídy odvozené od `ListView recordSpecification`, místo definování

podoby formuláře definují jen sloupce, které budou zobrazeny v tabulce. Na základě `recordSpecification` je vytvořen `Crm.data.ListViewStore`, jehož záznamy jsou opět pomocí `dataIndexu` *bindována* do tabulky zastoupené komponentou `Ext.grid.GridPanel`.

Crm.view.QuickView `QuickView` je rychlým náhledem na záznam. Může zobrazit například zkrácený detail při najetí myši nad referenci záznamu. Odvozené třídy definují `recordSpecification` a `templateMarkup` – šablonu HTML fragmentu pro zobrazení informací.

Crm.view.SingleCatalogView a Crm.view.MultiCatalogView `SingleCatalogView` a `MultiCatalogView` jsou pohledy sloužící pro výběr referencí. Jsou volány například z komponenty `ReferenceField`, nesoucí informaci o záznamu referencované entity. Odvozené třídy v obou případech definují `recordSpecification` a sloupce pro nadefinování tabulky. Pohledy se liší tím, že `SingleCatalogView` může vybrat pouze jednu hodnotu, zatímco z `MultiCatalogView` můžeme vybrat hodnot více.

Crm.view.DetailView `DetailView` je komplexní pohled na záznam entity. Obsahuje většinu informací o záznamu, mnohdy i kde je záznam dále referencován. Odvozené třídy definují `recordSpecification`, `customDetailContent` – konfiguraci samotného formuláře pro zobrazení detailu a funkci `getRecordCut` pomocí níž se získává název záznamu pro zobrazení.

5 Ukázka nasazení – RAYNET CRM

RAYNET CRM je primární aplikací využívající Crm frameworku, jehož vývoj je plně podřízen požadavkům RAYNET CRM. Přestože vývoj RAYNET CRM ještě nebyl dokončen, byl Crm framework ve starší verzi již úspěšně nasazen na komerční zakázce.

5.1 Zadání projektu

Cílem společnosti RAYNET je nabídnout RAYNET CRM jak v podobě *cloudové* aplikace, placené měsíčně dle počtu zakoupených licencí, tak ve formě CRM systému připraveného na míru podle požadavků zákazníka. Účelem cloudové verze je rozšířit povědomí o existenci a schopnostech CRM systémů mezi menší obchodní subjekty a nastínit jim tak výhody použití těchto systémů. Důležitá je také snadnost lokalizace pro zjednodušený vývoj jazykových mutací.

RAYNET CRM je plnohodnotnou aplikací CRM sloužící ke kompletní evidenci firemních obchodních vztahů. Cílem je evidovat nejen zakázky společnosti, ale také nabízet komplexní náhled na obchodní vztah s daným partnerem, či na vztahy s konkurencí. v CRM systému se evidují jak již hotové zakázky tak zájmy jednotlivých obchodních subjektů, poptávky, nabídky. V evidenci systému jsou jak samotné společnosti, tak osoby s danými společnostmi spojené, aktivity spojené s daným obchodním vztahem (korespondence, jednání). Uživatelům samotným aplikace umožňuje evidenci práce, plánování časového rozvrhu, fakturaci či přístup k firemním dokumentům.

Snahou společnosti RAYNET je nabídnout zákazníkům širší funkcionalitu než konkurenční produkty. RAYNET CRM nabízí například propojení se sociálními sítěmi – uživatelům RAYNET CRM bude například umožněno spravovat firemní stránky na Facebooku přímo z prostředí aplikace, či sledovat zdi evidovaných subjektů. Tato funcionalita je samozřejmě omezena právy uživatele k získávání jednotlivých údajů, lze tedy přistupovat pouze k těm informacím, které nejsou uživateli skryty.

5.2 Ukázka aplikace

5.2.1 Uživatelské prostředí aplikace

Nástěnka Po přihlášení do aplikace je uživateli zobrazena nástěnka[4], neboli *dashboard*. *Dashboard* RAYNET CRM používá upravené implementace `PortalColumnLayoutu`, který umožňuje do sloupců přidávat obsah v podobě *portletů* – malých aplikačních jednotek, které můžou sloužit např. jako poznámkový blok, štítky s připomínky, rychlý přehled zpráv, přehled aktuálně přihlášených uživatelů, atd. Jednotlivé *portlety* lze v rámci nástěnky různě přemisťovat, přidávat, odebírat či měnit jejich velikost. Zdrojové kódy jednotlivých *portletů* jsou dotahovány dynamicky v případě potřeby pomocí služby `PortletProvider`.

[4]Viz. [Obr.3] v příloze A

Rozložení *dashboardu* není ukládáno pomocí *cookies* jak je zvykem, ale implementuje `HttpStateProvider`, který se stará o ukládání rozložení na straně serveru. Ať už se uživatel přihlásí kdekoliv, vypadá jeho *dashboard* stejně.

Kalendář Důležitou součástí RAYNET CRM je kalendář[5] zobrazující naplánované aktivity. Aktivity jsou v rámci RAYNET CRM entitami, spojenými s činností zaměstnanců společnosti – jedná se o například o úkoly přidělené zaměstnanci, jednání která se mají uskutečnit s partnerem, kontaktování partnera – ať už telefonicky či e-mailem, atd. Pro přehled a plánování těchto aktivit slouží kalendář.

Samotná komponenta kalendáře je založena na komerčním rozšíření Ext-ensible Ext Calendar Pro (http://www.ext.ensible.com) a momentálně je v RAYNET CRM implementována pouze jako demonstrace možností a ověření specifikace požadavků.

Kalendářová komponenta ve finální podobě bude poskytovat několik časových pohledů (den, týden, měsíc), filtraci zobrazených záznamů pomocí různých filtrů (podle typu aktivity, podle určení).

5.2.2 Prvky Business logiky

`PersonListView`[6] je konkrétní implementací obecného `ListView`, mající za úkol zobrazit přehled osob evidovaných v systému. Osoby mohou být jak interní zaměstnanci, tak kontakty v rámci cizích firem. Označením záznamu `ListView` dojde k jeho zvýraznění a zobrazení zkráceného výpisu informací – `QuickView`.

`PersonInsertView`[7] implementuje obecné `InsertView` a slouží ke vkládání záznamu nové osoby do evidence systému. Pole Společnost je reprezentováno pomocí `ReferenceField`. Jeho rozkliknutím je vyvolán `CompanyCatalogView`[8], sloužící k výběru společnosti, kterou osoba zastupuje.

Po vytvoření nové osoby lze přejít buď zpět na přehled evidovaných osob, nebo na detail záznamu konkrétní osoby – `PersonDetailView`[9]. V detailu osoby jsou zobrazeny kontaktní údaje, vazby na všechny firmy, se kterými je osoba ve vztahu, poznámky a také přehled nejbližší aktivity a posledně vykonané aktivity – např. kdy má být osoba kontaktována a kdy byla kontaktována naposledy.

Většina systémem evidovaných entit umožňuje export údajů jak ve formě seznamu, tak ve formě detailu. k exportu se standardně nabízí formáty *.pdf* či *.xls*. Pro většinu entit je k dispozici také jedna či více tiskových sestav.

[5] Viz.[Obr.4] v příloze A
[6] Viz.[Obr.5] v příloze A
[7] Viz.[Obr.6] v příloze A
[8] Viz.[Obr.7] v příloze A
[9] Viz.[Obr.8] v příloze A

6 Závěr

Během vykonávání praxe ve společnosti RAYNET s.r.o. jsem se věnoval vývoji FE komponent založených na JavaScriptovém frameworku ExtJS. Na praxi jsem nastupoval bez předchozích znalostí jak problematiky JavaSctipových frameworků, tak bez znalosti pravidel týmové spolupráce.

Během praxe jsem získal potřebné znalosti pro vývoj RIA aplikací, aplikací propojených se sociálními sítěmi Facebook a Twitter, dále jsem rozšířil své znalosti HTML a CSS technologií a naučil se pracovat s nástrojem GIT pro správu verzí.

Z osobního hlediska považuji za velmi podstatné získání zkušeností s prací v týmu a s vývojem aplikací na profesionální úrovni.

7 Reference

[1] SHEA, Frederick; RAMSAY, Colin; BLADES, Steve. *Learning Ext JS,* Packt Publishing, 2008.

[2] GARCIA, Jesus. *Ext JS in Action,* Manning Publications, 2010.

[3] RAMON, Jorge. *Ext JS 3.0 Cookbook,* Packt Publishing, 2009.

[4] ASLESON, Ryan; SCHUTTA, Nathaniel. *Ajax: Vytváříme vysoce interaktivní webové aplikace,* Brno: Computer Press, a.s., 2006.

[5] LEHTINEN, Jarmo. *Aktivní CRM : Řízení vztahu se zákazníky,* Praha : Grada Publishing, 2007.

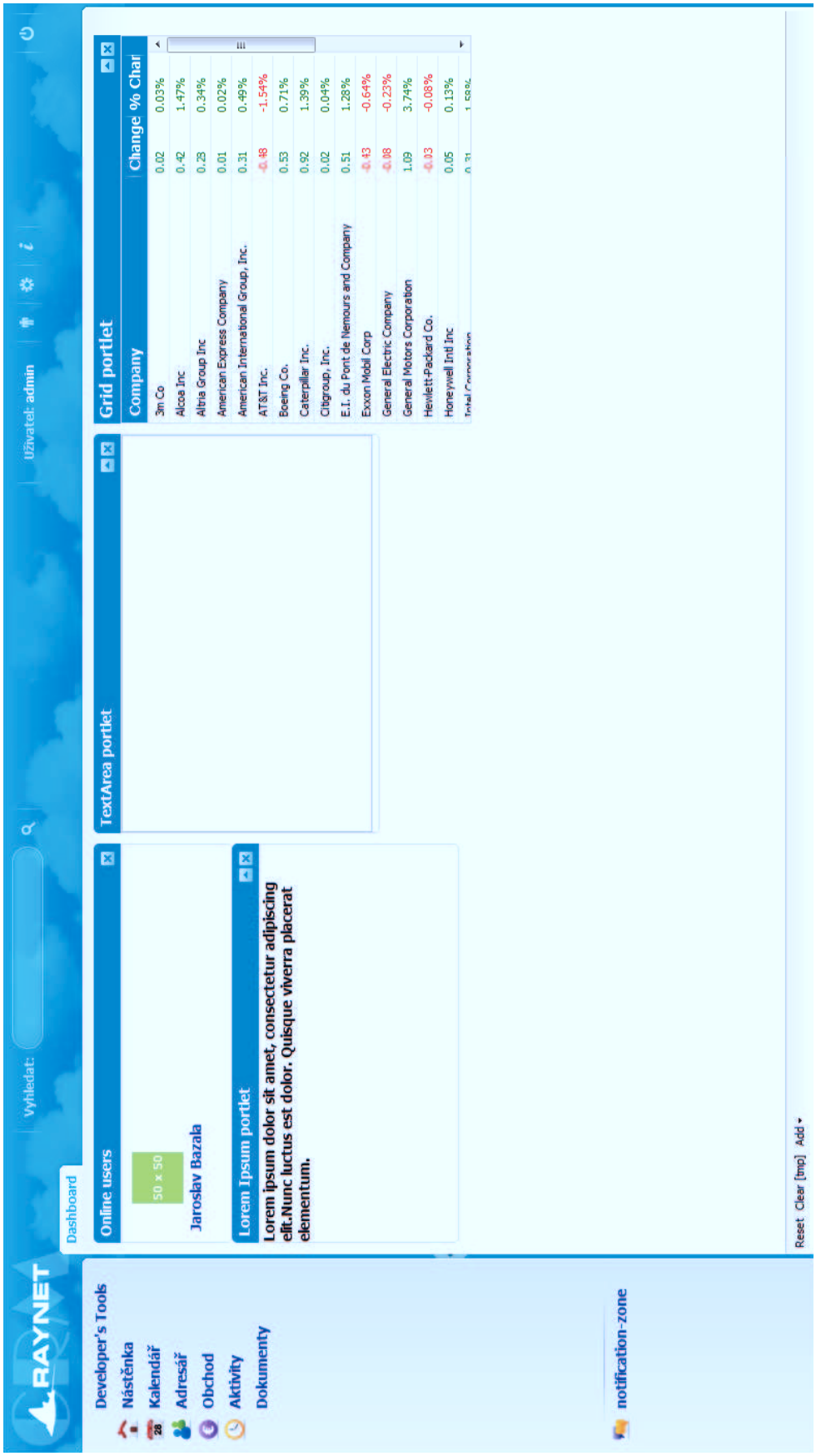
[6] PAVLÍČEK, Radek. *Přístupnost dynamických webových aplikací - úvod,* Zdroják.cz [online]. 6.5.2009 [cit. 2011-03-20]. Dostupné z WWW: http://zdrojak.root.cz/clanky/pristupnost-dynamickych-webovych-aplikaci-uvod/

[7] JUHOS, Miroslav. *Ext JS - javascriptový framework pro tvorbu RIA,* Zdroják.cz [online]. 17.2.2009 [cit. 2011-03-25]. Dostupné z WWW: http://zdrojak.root.cz/clanky/ext-js-javascriptovy-framework-pro-tvorbu-ria/

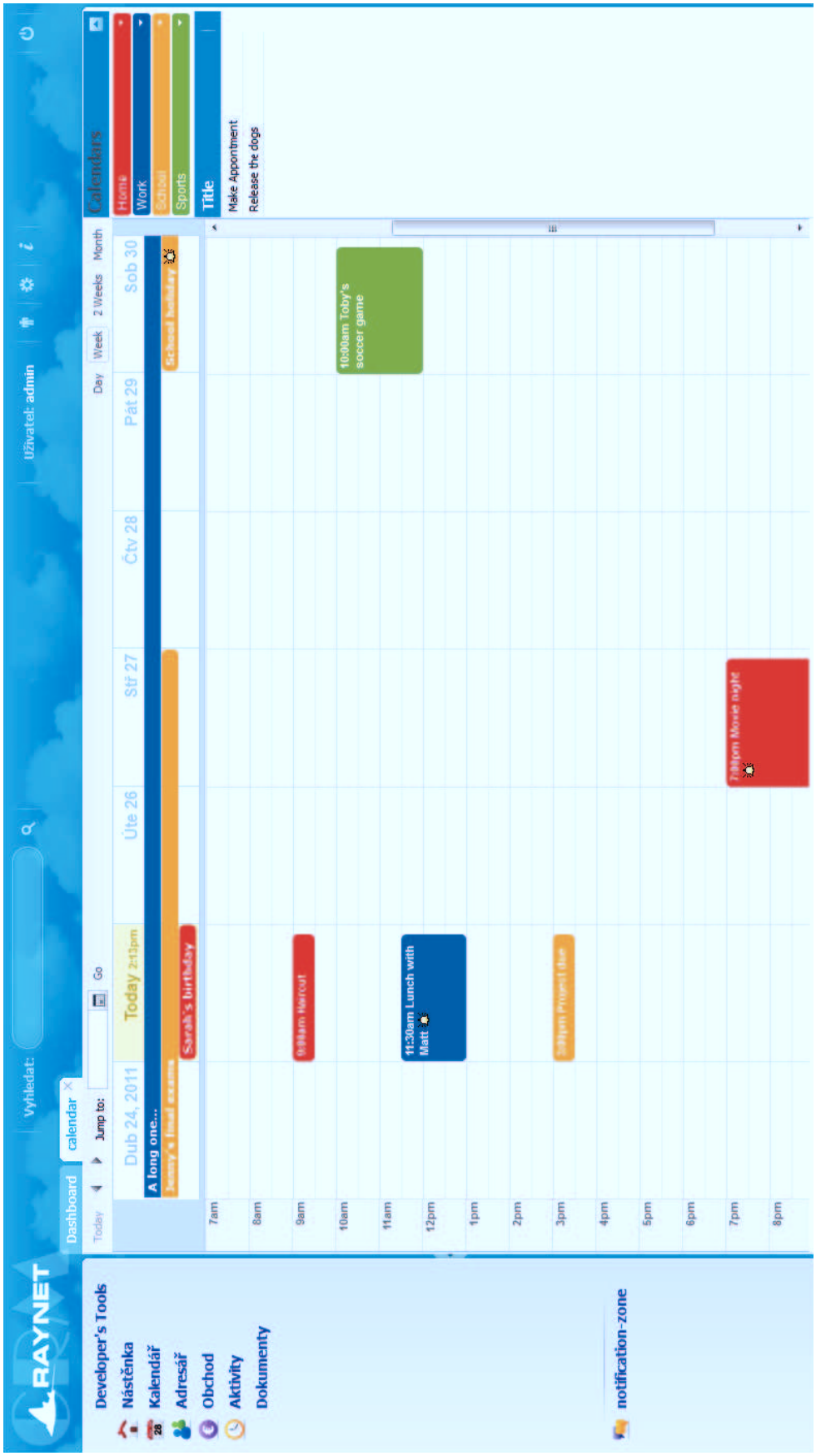
[8] PICHLÍK, Roman. *Rich Internet Application,* interval.cz [online]. 14.06.2005 [cit. 2011-03-20]. Dostupné z WWW: http://interval.cz/clanky/rich-internet-application/

A Obrazové ukázky RAYNET CRM

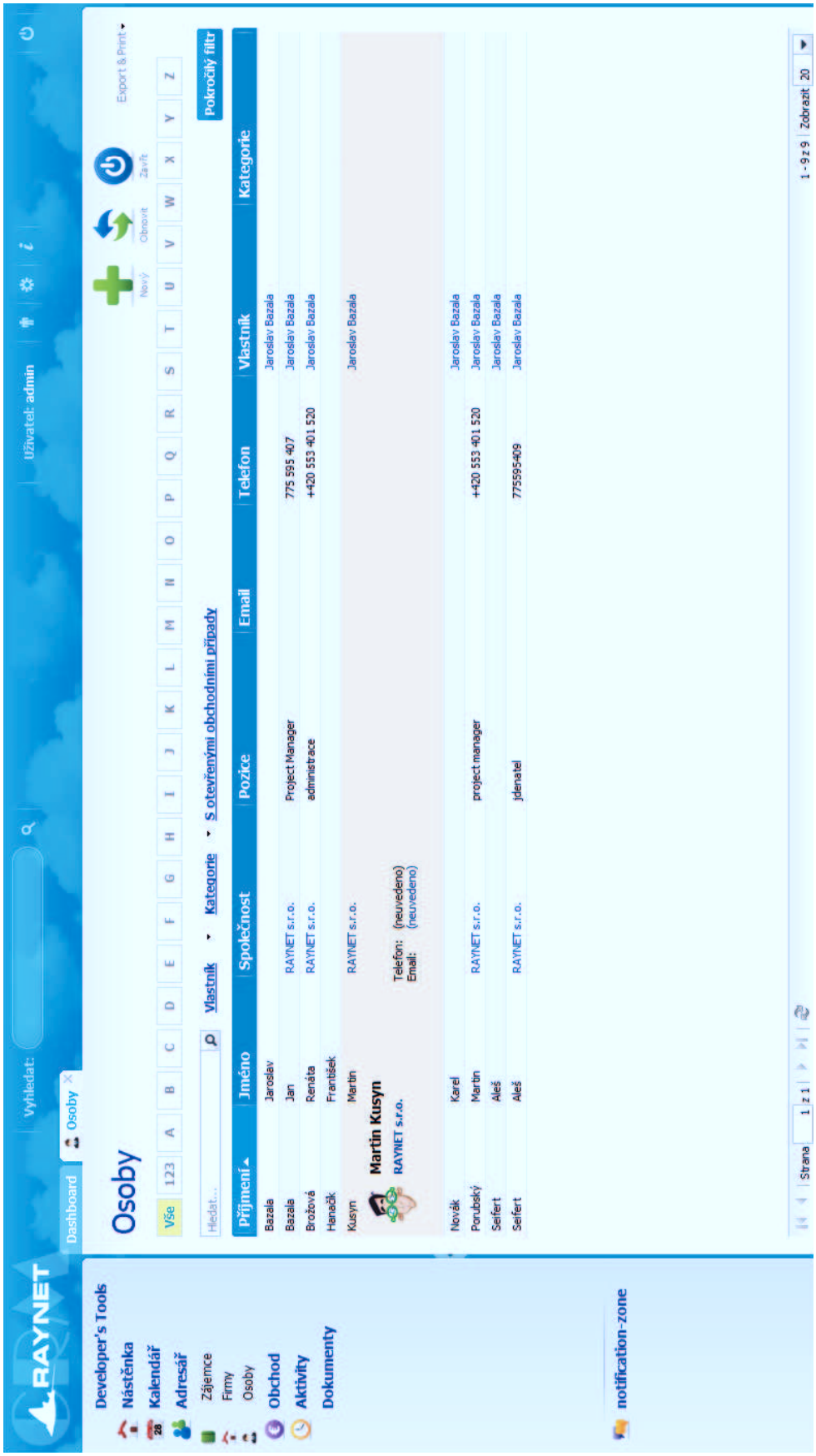
Tato příloha obsahuje obrazové ukázky důležitých komponent RAYNET CRM, které byly podrobněji rozebrány v kapitole Ukázka nasazení - RAYNET CRM



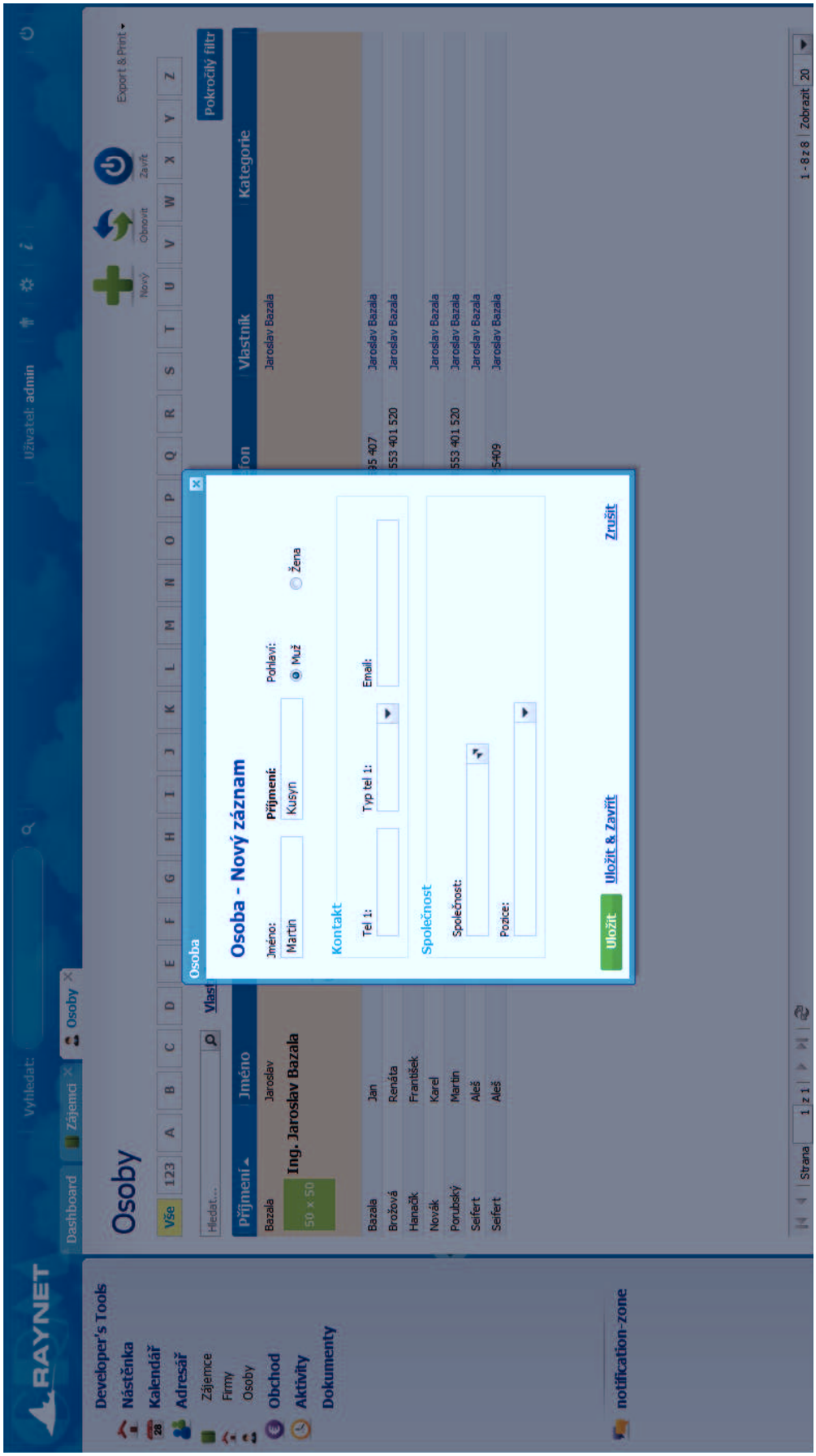
Obrázek 3: RAYNET CRM: Nástěnka uživatele



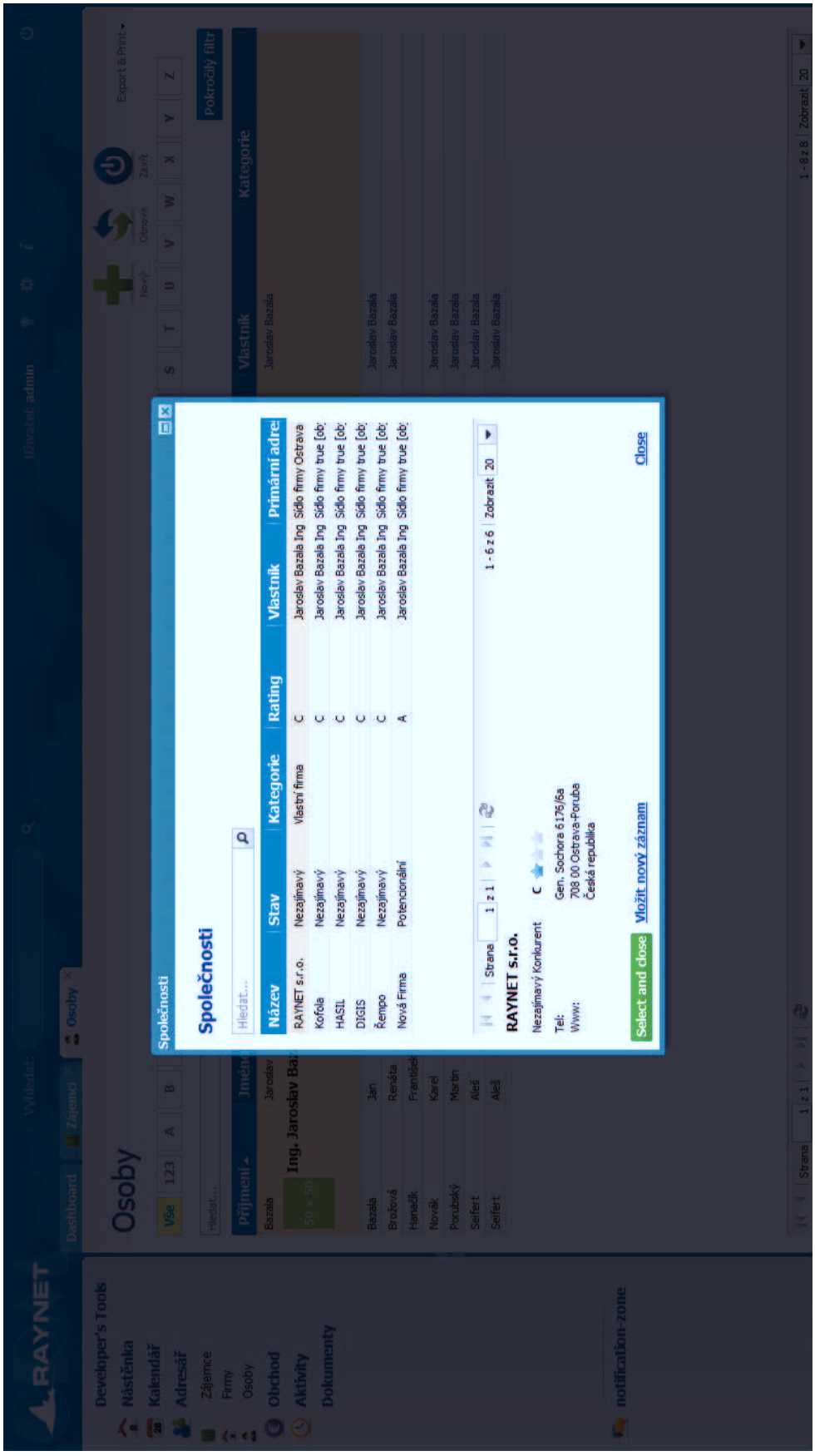
Obrázek 4: RAYNET CRM: Kalendář



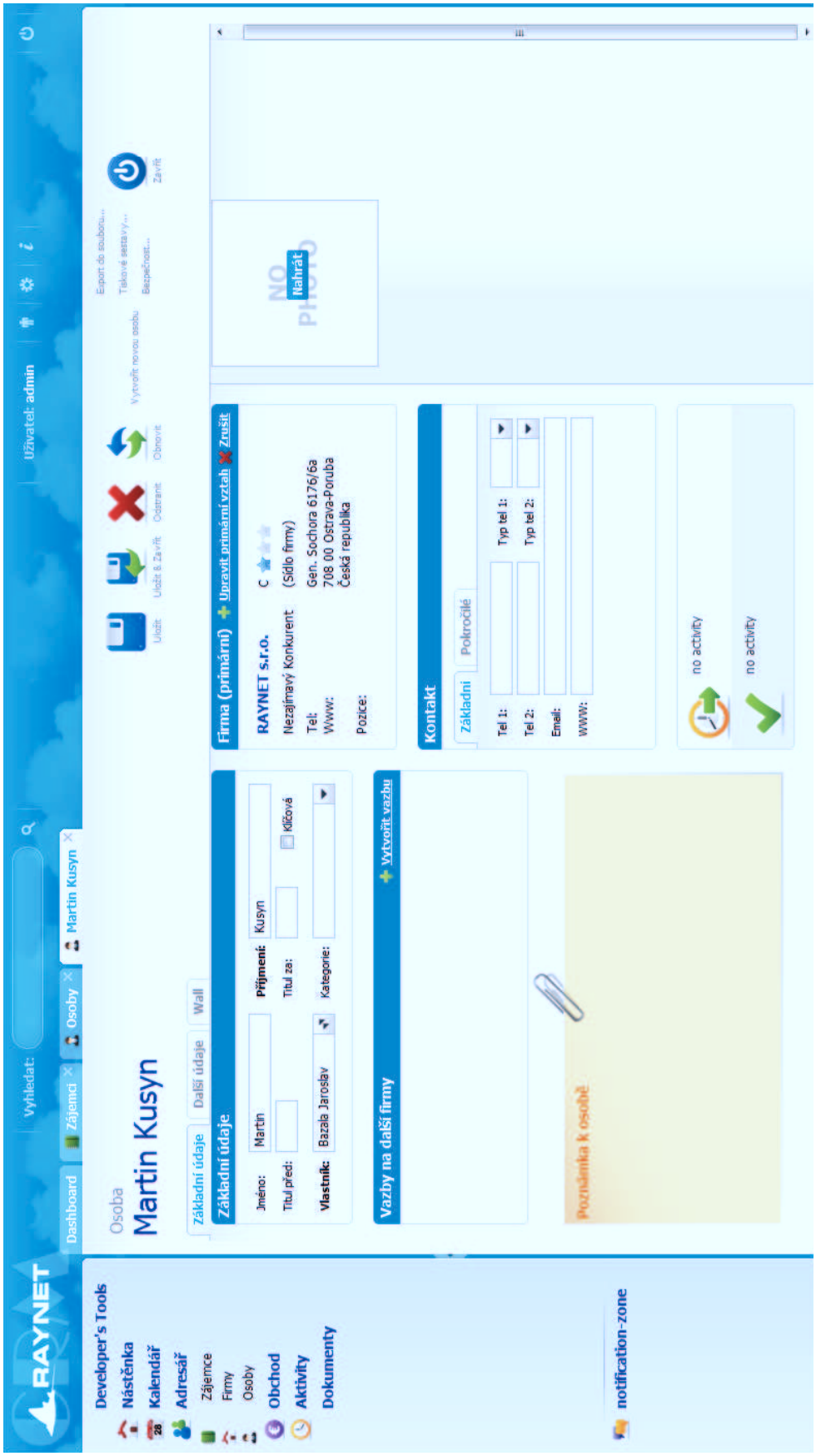
Obrázek 5: RAYNET CRM: Ukázka listView pro entitu OSOBA



Obrázek 6: RAYNET CRM: Ukázka insertView pro entitu OSOBA



Obrázek 7: RAYNET CRM: Ukázka catalogView pro entitu SPOLEČNOST



Obrázek 8: RAYNET CRM: Ukázka detailView pro entitu OSOBA

B Výpisy z kódu

Tato příloha obsahuje výpisy z kódu referencované v textu práce.

```
//Dojde k okamžitému vyvolání okna
onClick: alert('Upozornění'),

//Vytvoří se pointer na tuto funkci, která bude volána s definovaným parametrem
onClick: alert.createCallback('Upozornění'),

// Alternativní zápis
onClick: function(){
    alert('Upozornění');
}
```

Výpis 1: Příklad použití callback funkce v ExtJS

```
Ext.Ajax.request({
    url: 'foo.php',
    success: function(connection, response, options){
    //zpracování výsledků
 },
    failure: function(connection, response, options){
    //ošetření chyby
},
    headers: {
        'my-header': 'foo'
    },
    params: { foo: 'bar' }
});
```

Výpis 2: Příklad vyvolání AJAX requestu v ExtJS

```
//vytvoření instance tlačítka
var btn = new Ext.Button({
        title : 'Click me too',
       handler: alert.createCallback('Clicked'),
       width: 100
   });

//vytvoření instance panelu
var p = new Ext.Panel({
    title : 'Container Panel',
    //Nastavení defaultního typu
   defaultType: 'panel',
    //Nastavení defaultních hodnot
   defaults: {
       widht: 200,
   },
   items: [{
       //Přidání komponenty, která bude splňovat defaultní nastavení – panel široký 200px
        title : 'Panel 1',
       html: 'Lorem Ipsum Sit Amet'
   },{
        title : 'Panel 2',
       html: 'Lorem Ipsum Sit Amet'
   },{
       //Tato komponenta bude vytvořena jako tlačítko, z˜defaultních nastavení přebírá šířku 200px
       xtype: 'button',
        title : 'Click me',
       handler: alert.createCallback('Clicked')
   },
    //přidání již nadefinované komponenty
   btn
   ]
})
```

Výpis 3: Příklad vytvoření kontejneru a použití xtype

C Příloha na CD

Na přiloženém CD se krom elektronické verze této práce nachází také ukázková aplikace, prezentující možnosti RIA aplikací. Základ této aplikace pochází od mého kolegy Bc. Martina Stříže, který tuto aplikaci naprogramoval pro svou diplomovou práci zabývající se převážně BE stranou RIA aplikací. Ukázková aplikace nicméně obsahuje také FE prezentující některé entitní pohledy zmíněné v rámci Crm rozšíření ExtJS frameworku.

Z FE původní aplikace jsem odebral série pohledů pro jednotlivé entity a pro demonstraci jsem ponechal jsem pouze pohledy pro správu uživatelů aplikace. Dále jsem FE aplikace doplnil o zjednodušené ukázky řešení úkolů, které mi byly v rámci praxe zadány.

Ukázková aplikace samozřejmě není plnohodnotnou verzí RAYNET CRM, jedná se pouze o velmi zjednodušenou aplikaci reprezentující pouze velmi základní možnosti RAYNET CRM.

Více detailů o ukázkové aplikaci se spolu s návodem na spuštění nachází na CD.

Poděkování Tímto bych rád oficiálně poděkoval Bc. Martinu Střížovi za poskytnutí aplikace, z níž ukázka vychází.